2007

Quasi group based crypto-system

Maruti Venkat Kartik Satti
Louisiana State University and Agricultural and Mechanical College

QUASI GROUP BASED CRYPTO-SYSTEM

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

In

The Department of Electrical Engineering

By
Maruti Venkat Kartik Satti
B.Tech, Gokaraju Rangaraju Institute of Engineering and Technology, 2005
Hyderabad, India
December 2007

Acknowledgements

This work would not have been possible without the constant encouragement and support I received from Dr. Subhash Kak, my advisor and mentor. I would like to express my sincere thanks to him. I would also like to offer my gratitude to Dr. Xue-Bin Liang and Dr. Hsiao-Chun Wu for serving on my thesis advisory committee.

I would like to thank Dr. Markovski and for his help and co-operation. I would also like to thank all my friends who have helped me all through my stay at LSU. I express my grateful appreciation to my parents for their love, encouragement and support. Through them I have learned that perseverance is the key to success.

TABLE OF CONTENTS

Abstract

For electronic commerce and other applications it is required to encrypt data that is transmitted over an unsecured channel. The data is encrypted/randomized using a key. Algorithms such as DES and ECC randomize the data such that un-authorized user cannot decrypt it .This thesis presents a practical implementation of a quasi group based multilevel, indexed scrambling transformation for use in signal encryption. Results of experiments with text and speech scrambling are presented. It is shown that the quasi group transformation maximizes the entropy at the output, which is desirable for a good system. This system provides extremely large group of keys that ensures enhanced security. It can work in either the chain mode or the block mode. Block mode is more tolerant to errors compared to the chain mode.

Chapter 1: Purpose and Technical Background

1.1 Introduction

Security is of fundamental importance in digital communication. . The system should be secure against brute force attacks and impersonation by the eavesdropper. Encryption is the general name given to coding the sensitive data such that only legitimate user/authority can understand it.

The idea of encryption has been around for about over 2000 years. Julius Caesar is credited with developing a substitution cipher that is named after him now. The Kama sutra cipher [20] is yet another substitution cipher that has been used in India for nearly two millennia.

With the advent of micro-technology, computers have become more powerful. Simple substitution ciphers are no longer safe for encoding sensitive personal information. This need for a better encoding system led to the development of symmetric and asymmetric key cryptosystem. In symmetric cryptosystem one key is used to encrypt the data and the same key is used to decrypt it. This system can be broken easily by brute force if the key is not sufficiently large.

Public Key Encryption, or asymmetric key encryption, is much more secure than symmetric key encryption for the purposes of e-commerce. The main difference in Public Key Encryption was the introduction of the second key - which makes it very difficult to crack by increasing the number of possibilities. This scheme relies on two keys, one of which is public and the other is private. If you have one key, the other key cannot be inferred from it. The public key is published and any user who wants to connect to communicate with a business has to use the public key to encode his message. At the destination the intended business decodes it using their private key

To avoid any foul play this system needs central trusted authority to distribute “Digital Certificates” to valid users/business.

One of the most famous encryption systems, RSA, is based on asymmetric key concept. In this cryptosystem if Alice wants to communicate with Bob, she transmits her public key (n,e) to Bob and keeps the private key secret. Bob then wishes to send message **m** to Alice.

He first turns **M** into a number m < n. He then computes the cipher text 'C' corresponding to:

$$c = m^e \mod n$$

This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice. Alice can recover 'm' from' c' by using her private key 'd' by the following computation:

$$m = c^d \mod n.$$

Thus she can recover the original message 'm'

It can be observed that the strength of the RSA cipher depends on the size of 'n' and if it can be factorized the RSA cryptosystem is no longer secure. In 1994, Peter Shor published Shor's algorithm, showing that a quantum computer could in principle do the factorization in polynomial time. However Kak in [14], [15] and [16] proposed that the idea of quantum computing is not very feasible. Consider the discrete possibility of quantum computers being possible in the near future, and then the RSA cryptosystem would be obsolete. Thus one way of making a crypto-system secure is to increase the number of keys and that what the proposed crypto-system does. As a matter of fact the Quasi group crypto-system can generate more keys than ECC even when we consider small orders (order of the Quasi group).

On the other hand one might say that the proposed (MIQE) Cryptosystem is similar to DES. It is agreeable that just like DES and Triple DES this is some sort of scrambling technique. However if the

key is kept secret for the time of transmission it would be nearly impossible to extract the encrypted data. It is found that the number of isotopes increase with the increase in the order of n!(n-1)! where n is the order of the quasi group.

1.2 Technical Background

Permutation transformations are a basic operation in many scrambling and encryption systems. Many early ciphers for data and speech security [2], [3], [6], [8], [12],[13] were based on permutation transformations, and message authentication systems also use such transformations. It was shown in an earlier study by N.S. Jayant and one of the authors of the present paper [6] that a simple permutation of samples does very well in destroying intelligibility of speech, but for further protection against brute force attack it is essential to increase the number of possible keys. Quasi groups (or Latin squares) provide a powerful method for generating a larger set of permutation transformations by permuting not only the samples but also transforming the amplitudes themselves across their range [11]. By doing this, they provide an immensely large number of keys, even for small alphabets. Therefore, quasi group based ciphers can have a variety of applications, and in some cases can be competitive to number theory based systems in terms of the difficulty they offer to brute force attacks.

This thesis presents an implementation of quasi group-based permutations that has very good encryption properties and, therefore, it has potential uses in symmetric cryptography as in design of message authentication and hash functions as well as in speech scrambling. If the purpose of the scrambler is to maximize the entropy at the output, then we see that the quasi group based system accomplishes this successfully, even for input data that is constant. The immense complexity associated with the task of finding the scrambling transformation ensures the effectiveness of the encryption process.

Quasi group based scrambling has been proposed before, but the earlier techniques such as those of Markovski and collaborators [4],[5],[9],[10] are insecure because their security is predicated only on

the initial multiplier element that is like the seed of a random number generator. Here we propose a new multilevel indexed implementation that allows several layers of permutations to be carried out in a manner that is not only flexible but which enhances security. It distributes raw data stream based upon the indices and the order of the quasi group under consideration. This unique key which consists of the index numbers and the matrix orders *r* and *s* (associated with the permutations that are performed) is kept secret and is transmitted by the trusted authority in a way that the keys do not repeat till a pre-specified network time expires.

The output of the proposed encryptor is dependent upon the index numbers and the orders of the matrix transformations that are sent by the trusted authority. The encryption is also dependent on six multiplier elements that are generated by a secret algorithm based on the index numbers, the order of the matrices under consideration and *nonce* (random number generated by trusted authority). This *key* s updated by the network on a regular basis (once in every time interval that is far less that T, the time needed to use brute force to decrypt the key sent by the trusted authority). Figure 1 illustrates the quasi group encryptor .The module here takes the raw data stream and randomizes it based on the encryption key (the encryption key). We show later that the output data has desirable autocorrelation properties. We have applied this multilevel scrambling approach to text and speech problems with good effect.

This thesis is organized as follows:

In chapter 2 we introduce the reader to basics of Latin squares and quasi groups. Chapter 3 introduces quasi group signal processing in MIQE. Chapter 4 describes the operation of the multilevel indexed quasi group encryption (MIQE) system. Chapter 6 deals with MEG1 and FG1 and Chapter 7 discusses the randomization properties of the cryptosystem .Chapter 8 presents a quasi group voice scrambling system. Chapter 9 deals with the security aspects and explains the relative strength of the system. The conclusion is in Chapter 10.

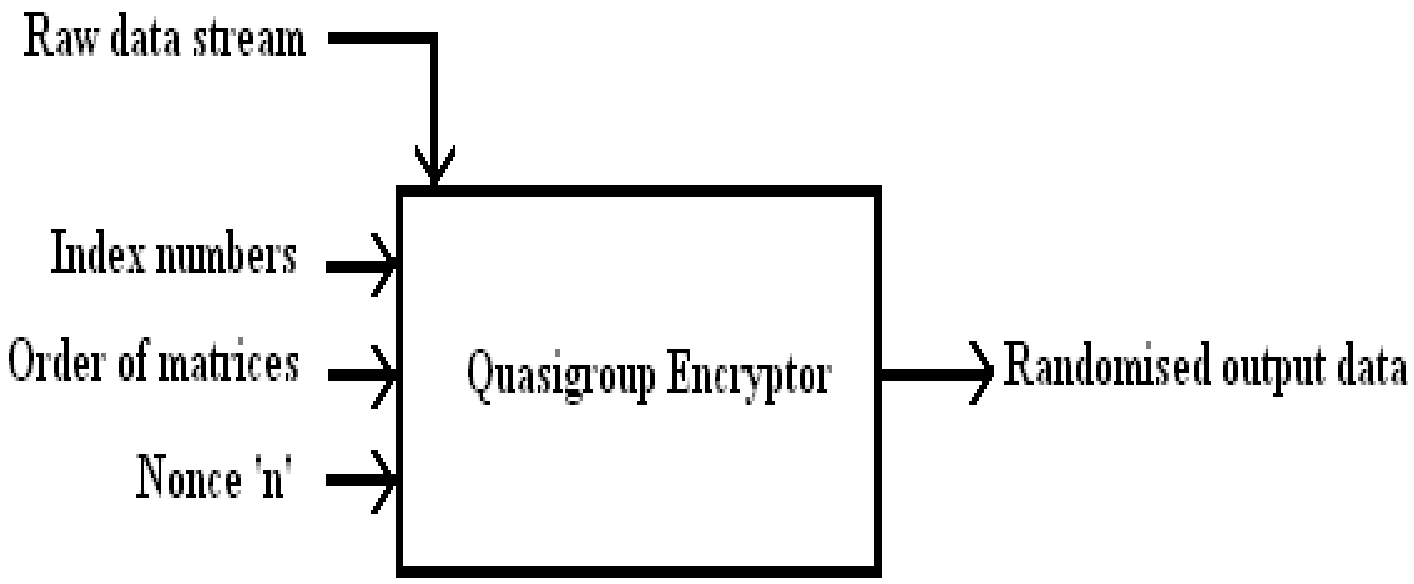


Figure1: Quasi group encryptor

Chapter 2: Latin Squares and Quasi Groups

2.1 Latin Squares and Isotopes

A Latin square of order 'n' is an 'n' by 'n' array in which each of the 'n^{2}' cells contains a symbol from an alphabet of size n, such that each symbol in the alphabet occurs just once in each row and once in each column. We build a matrix such that the row or column consists of elements from 1 through 'n'.If we permute in any way the rows, or the columns, or the symbols, of a Latin square, the result is still a Latin square. We say that two Latin squares L and L_0 are isotopic if L_0 can be obtained from L by performing certain row or column permutations or substitution or all three on it. Based upon the order we can have several isotopes of the same fundamental quasi group. The number of isotopes can be give be n!(n-1)! Which is tremendously large for large values of n, for example for n=5 we have a total of 161280 Latin squares.

In the proposed implementation we generate the fundamental Latin square based on the priori criteria and then generate the isotopes based on the same. Since there can be many isotopes for any given Latin square, we need to set a limit on the number of isotopes that are to be generated. Since it the number of isotopes generated is a compromise between the security and the space available; it is up to the network provider and the user to come to a common agreement. It is imperative that the length of the index vector is less than the number of isotopes generated.

2.2 Quasi Group Definitions and Theory

A quasi group Q may be defined [1] as a group of elements (1,2,3,… n) along with a multiplication operator such that for its elements x and y there exists a unique solution z, also belonging to Q, such that the following two conditions are obeyed:

1. x*a=z (1)

2. y*b=z

A quasi group may be defined alternatively as a binary system (Q,*) satisfying the two conditions:

1. For any a, b belonging to Q there exists a unique x belonging to Q such that a*x=b
2. For any a, b belonging to Q there exists a unique y belonging to Q such that y*a=b (2)

The multiplication table of a finite quasi group is a Latin square. In effect it means that a if we make a square matrix with n symbols such that a symbol does not repeat itself in either the row or the column then we get a Latin square. Now if we index the Latin square we get a Quasi group. If we consider the row index as 'q' and the column index as 'p' then the product of 'p' and 'q' would give 'a' (an entry in the table correspond to row index 'q' and column index 'p'). Thus 'a' denotes the product of 'q' and 'p'. This is not to be confused with the general group theory. We however call it a group because the product of any two symbols within the quasi group results in an element that lies with the symbol set that constitutes the group.

Example 1: An example of a quasi group Q of order 5 with elements (1, 2, 3, 4, 5) is given by the element multiplications of Table 1.

A multiplication operator in a quasi group behaves as a mapping between the row and the column indices.
For example if x = 2 and a = 3, the resulting 'z' can be determined by looking up the element having the row index of 2 and the column index of 3 in Table 1. We get the value of z as 5.

Table 1: Multiplication table for a quasi group, n = 5

*	1	2	3	4	5
1	3	4	2	5	1
2	4	1	5	2	3
3	2	3	1	4	5
4	1	5	4	3	2
5	5	2	3	1	4

Chapter 3: MIQE Encryption and Decryption

3.1 Quasi Group Signal Processing in MIQE

This section introduces the notation in the description of our multilevel indexed quasi group encryption (MIQE) system.

Input data: $d_1, d_2, d_3, \ldots d_n$

Output data: $e_1, e_2, e_3, \ldots e_n$

Transformation matrices: R, S

Multiplier elements: $q_1, q_2, q_3, \ldots q_n$

Indices: $I_1, I_2, I_3, \ldots I_n$

The encryptor is represented by QE (stands for Quasi-Encryptor) and the decryptor is represented as QD (stands for Quasi-Decryptor) . The transformation matrices R,S are the fundamental matrices for which the receiver generates the isotopes based upon an agreed criterion. Even the number of isotopes generated is dependent on these criteria. The multiplier elements are generated by the MEG1 module inside the receiver. Each entry in the hidden key vector is used as a multiplier element with the corresponding isotope index in the index vector. The output data is obtained after several levels of encryption (we have implemented six layer encryption). However the number of isotopes used for encryption depends on the level of security needed.

3.2 Encryption

Since the quasi group encryptor performs an operation on the input data and produces a randomized output sequence it can be considered as some sort of a function that operates on data. In this thesis we have only discussed about encryption of a data sequence. However one can also use this cipher for block

encryption. The block encryption scheme has the advantage of low error rate while the stream cipher implementation has the advantage of being simple to implement.

The mathematical equation used for encryption is given by:

$$E_a(a_1,a_2,a_3,\ldots,a_n) = b_1,b_2,b_3,\ldots b_n \quad (3)$$

Where E stands for the encryption function and output sequence is defined by:

$b_1=a*a_1$

$b_i=b_{i-1}*a_i$

Where *i* increments from 2 to the number of elements that have to be encrypted, and *a* is the *leader* or the *hidden key*. Equation (3) describes a typical single level quasi group encryptor.

We now illustrate the working of equation (3) with the help of the illustration of Figure 2, for which the value of a=2. This equation maps the initial input data vector $(a_1,a_2,a_3,a_4,a_5,a_6) = (2,4,1,2,3,3)$ into the vector $(b_1,b_2,b_3,b_4,b_5,b_6)$ for the case of the quasi group of Example 1 with the multiplication relationship of Table 1.

The following steps are used during the process of encryption:

$b_1=a*a_1=2*2=1$

$b_2=b_1*a_2=1*4=1$

$b_3=b_2*a_3=4*1=4$

$b_4=b_3*a_4=4*2=5$

$b_5=b_4*a_5=5*3=1$

$b_6=b_5*a_6=1*3=2$

The sequence thus obtained (1,1,4,5,1,2) is then given as an input to another level of the encryptor. This process is repeated several times. This multiple levels of mapping ensure that the resemblance of the output data to that of the input data is minimized, making it harder for the eavesdropper to decrypt the data until he has sufficient information regarding the number of times the mapping was done by the sender.

In a variation to this approach, the multiplier element is varied, and generated by a special algorithm called MEG1 that generates the multiplier elements based on the index numbers, nonce, and r and s. This variant implementation may be given by the following equations:

$$E_{h1,h2,h2\ldots hn}\ (a_1,a_2,a_3,a_4\ldots a_n)=e_1,e_2,e_3,\ldots e_n \qquad (4)$$

where

$e_1=a*a_1$ and $e_i=e_{i-1}*a_i$

In the above equation the incoming stream of data is first mapped using the first multiplier element h_1 then the resultant steam is mapped considering the second multiplier element h_2 , and this process continues till all the multiplier elements are exhausted.

Consider Equation 5, the vector $(h_1,h_2,h_3,\ldots\ h_n)$ consists of all the multiplier elements .In this approach this encryption key is transmitted along with the quasi group (this key is itself encapsulated by another

layer of encryption). It is understood that in the above two approaches another reliable encryption algorithm is required to preserve the secrecy of the encryption

$b_1=h_1*a_1$; $b_2=b_1*a_2$;… $b_n=b_{n-1}*a_n$

(5)

$c_1=h_2*b_1$; $c_2=c_1*b_2$;... $c_n=c_{n-1}*b_n$

.

.

$e_1=h_n*s_1$; $e_2=e_1*s_2$;…$e_n=e_{n-1}*s_n$

Moreover, one must transmit information regarding the quasi groups being used for encryption, and this constitutes one of the main drawbacks of the above approach.

The third approach is the index based approach where the given data is encrypted through several levels by the encryption. Let us consider that this changed order encryptor is given an input of all 1s (as illustrated in Figure 3). We see that after the second level encryption the input vector is mapped to the sequence that has symbols ranging from 1 to the order of the second matrix. Therefore, if we have an index key which references the matrices stored in the memory of the reception device, the eavesdropper would not know which matrix is stored at a given index. To further improve the efficiency of this encryptor we can include another function that arranges the quasi groups based upon a nonce that makes the output more independent of the input. At any given time the output of the encryptor is different even if the same set of indices is supplied to the algorithm.

Thus encryption in MIQE is represented by

$$QE_{h_1,h_2,\ldots h_n}^{I_r,I_s}(a_1,a_2,a_3\ldots a_n)=e_1,e_2,e_3,\ldots e_n \quad (6)$$

where $(a_1,a_2,a_3,\ldots a_{n)}$ is the input vector and $(e_1,e_2,e_3,\ldots e_{n)}$ is the output vector, I_r and I_s are indices corresponding to the order of the quasi groups. The vector $(h_1,h_2,h_3\ldots h_n)$ is called the *hidden key* or the *secret key*. It is the output of the MEG-1 algorithm.

In *hidden key* vector the first half entries are generally used as multiplier elements with the first half index numbers in the *Index vector* (index in short refers to the isotope of the correspond quasi group 'r' or 's') and the second half are used as multiplier elements with the second index numbers in the *Index vector*.

In this approach even if the illegitimate user has access to the index numbers he need to know the inbuilt protocol MEG1 in order to decipher the encoded data. This scheme is scalable because the network administrator is given the option of setting the security level by assigning the number of isotopes and the size of the quasi groups used for encryption. Even if the eavesdropper has the idea of the index numbers and group orders; he still has to generate the isotopes based on a priori criteria. It would be very difficult to break this cipher using brute force even with the knowledge of the indices and group orders because these refer to the isotopes that are present in the database of a legitimate user.

From figure 3 it can be observed that after we obtain b1,b2,… b10 from the first quasi group operation we perform a second encryption with the help of a quasi group whose order is greater than the first. By doing this we further mask the original data. It must be noted that the actual implementation consists of three encryption in layer one (with 3 isotopes of the first quasi group) followed by three encryptions in layer two (with 3 isotopes of the second quasi group). Every entry in the *hidden key*

Vector corresponds to the multiplier element of the corresponding isotope as referenced by the index vector.

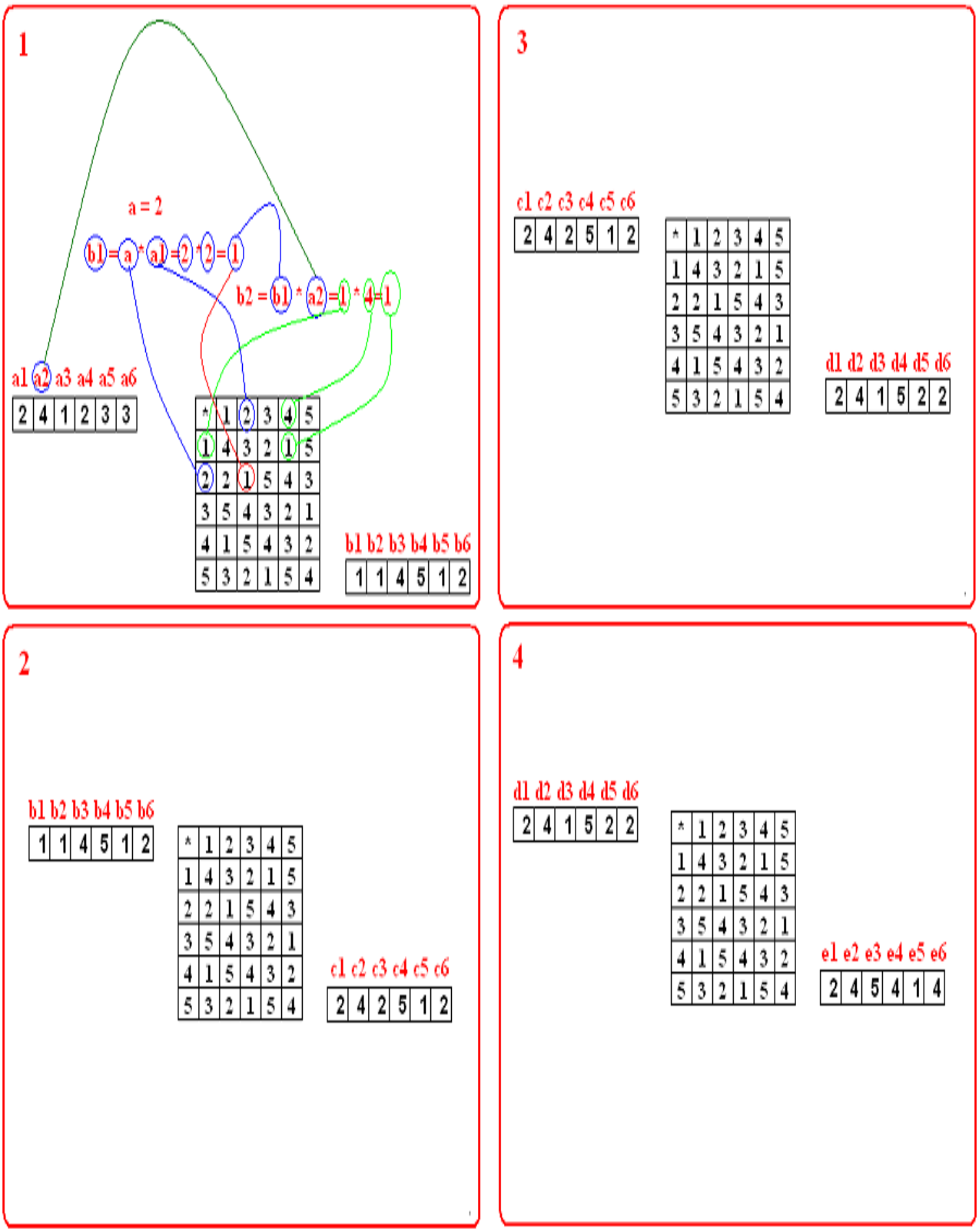


Figure 2: Quasi group mapping using an order 5 quasi group

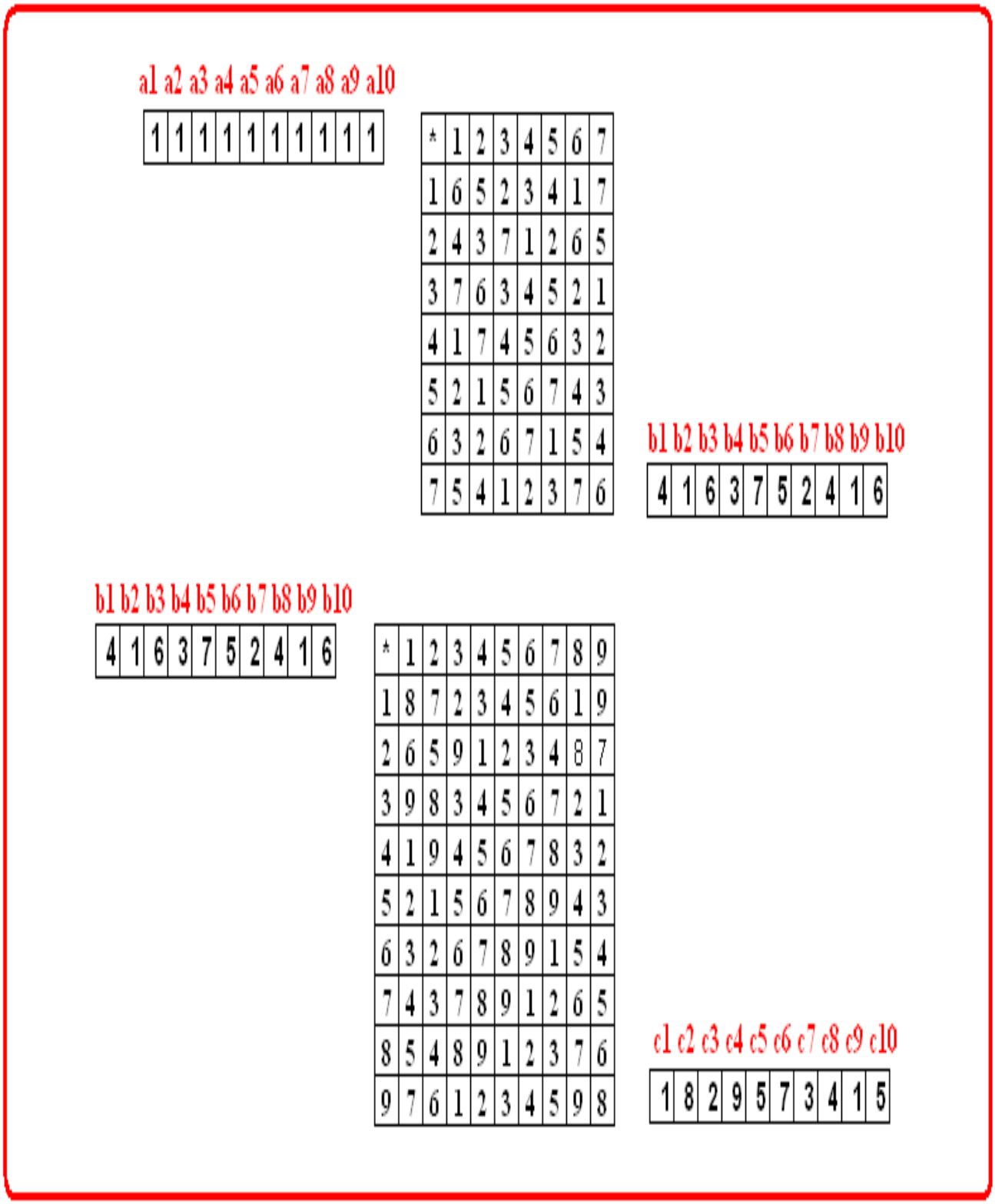


Figure 3: Encryption using 2 different groups of different order

3.3 Decryption

This process is similar to encryption. First, the generation of the inverse matrix will be described. For this one needs to understand the left inverse '\' I used for the quasi group decryption (Figure 4 illustrates the encryption and decryption of data).

The basic equation for encryption is as below:

$$D(a_1,a_2,a_3,\ldots a_n)= e_1,e_2,e_3,\ldots e_n \quad (7)$$

Where

e1=a\a1 and ei=ai-1\ai

To perform the process of decryption we need to first generate the inverse matrix of a given quasi group and execute mapping procedure as described in the previous section (Figure 2). This must be done using (7) rather than (6).

The decryptor for a multilevel indexed based algorithm cam be defined as follows:

$$QD^{I_r,I_s}_{h_n,h_{n-1},..h_1}(e_1,e_2,e_3,...e_n) = a_1,a_2,a_3,...a_n \quad (8)$$

The method of the MIQE decryptor is similar to the MIQE encryptor as shown in the next section.

1. The elements of the quasi group (marked as 1 in Figure 4) are labeled as *w*, the indices along the horizontal are labeled *v* and the indices along the vertical are labeled *u*.
2. The elements of the inverse (left-inverse) of quasi group (labeled as 2 in Figure 4) are labeled as *v*, the indices along the horizontal are labeled *w* and the indices along the vertical are labeled u^{-1}.

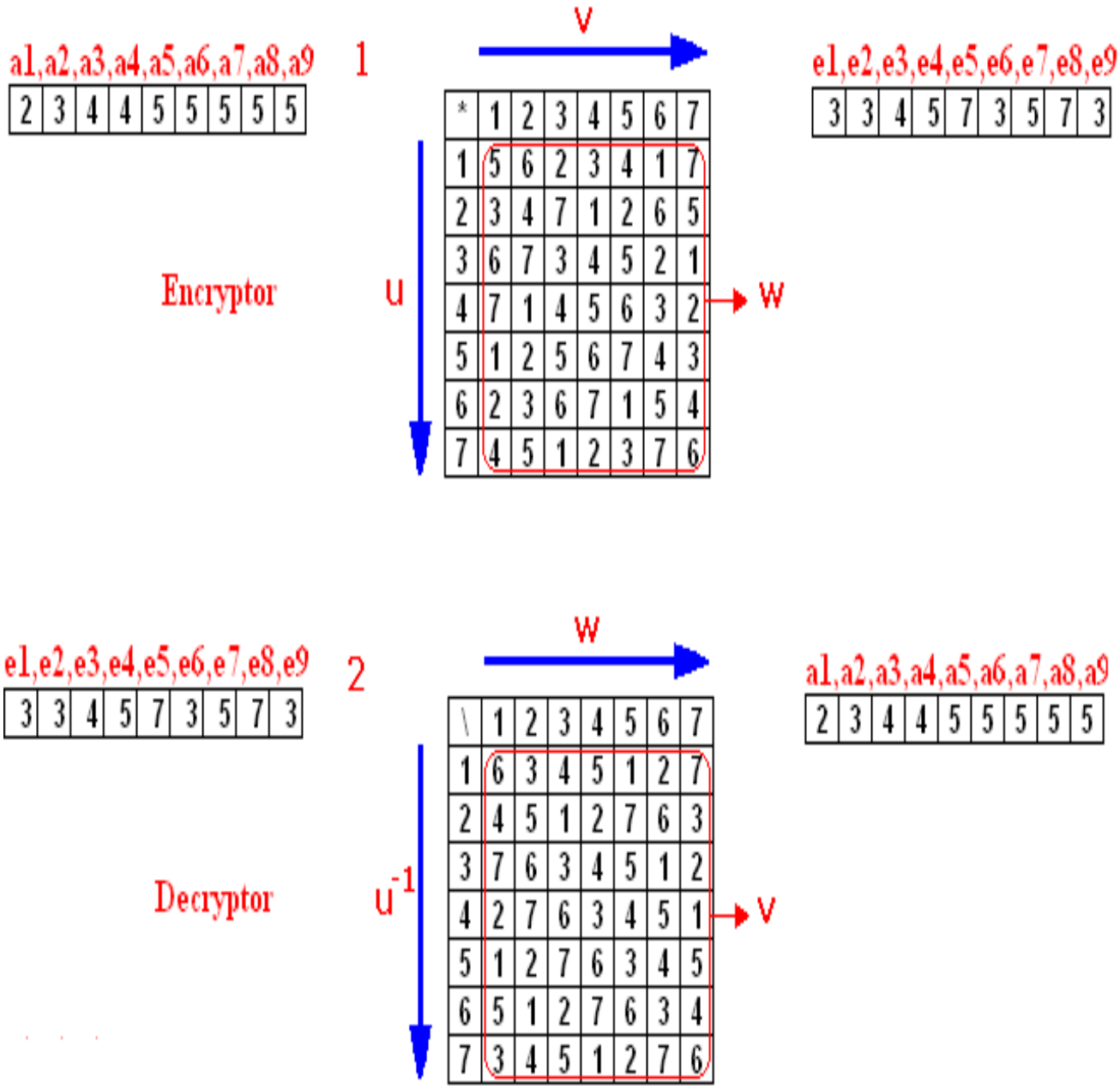


Figure 4: Determination of left division and the compete process of encryption and decryption

3.4 How to Determine the Left Inverse of a Quasi Group

By executing the following algorithm, one can generate the left inverse of a given quasi group:

1. Scan the row of the quasi group
2. Locate an element *i* (start the value of *i* from 1) and note the value of *v* in *the* corresponding location of the *in*verse matrix.

3. Increment *i*.
4. Go to step 1.
5. This process continues till all the elements in the row are exhausted.
6. Then go to the next row and repeat steps 1 through 3.

Chapter 4: MIQE Design

4.1 MIQE Encryptor

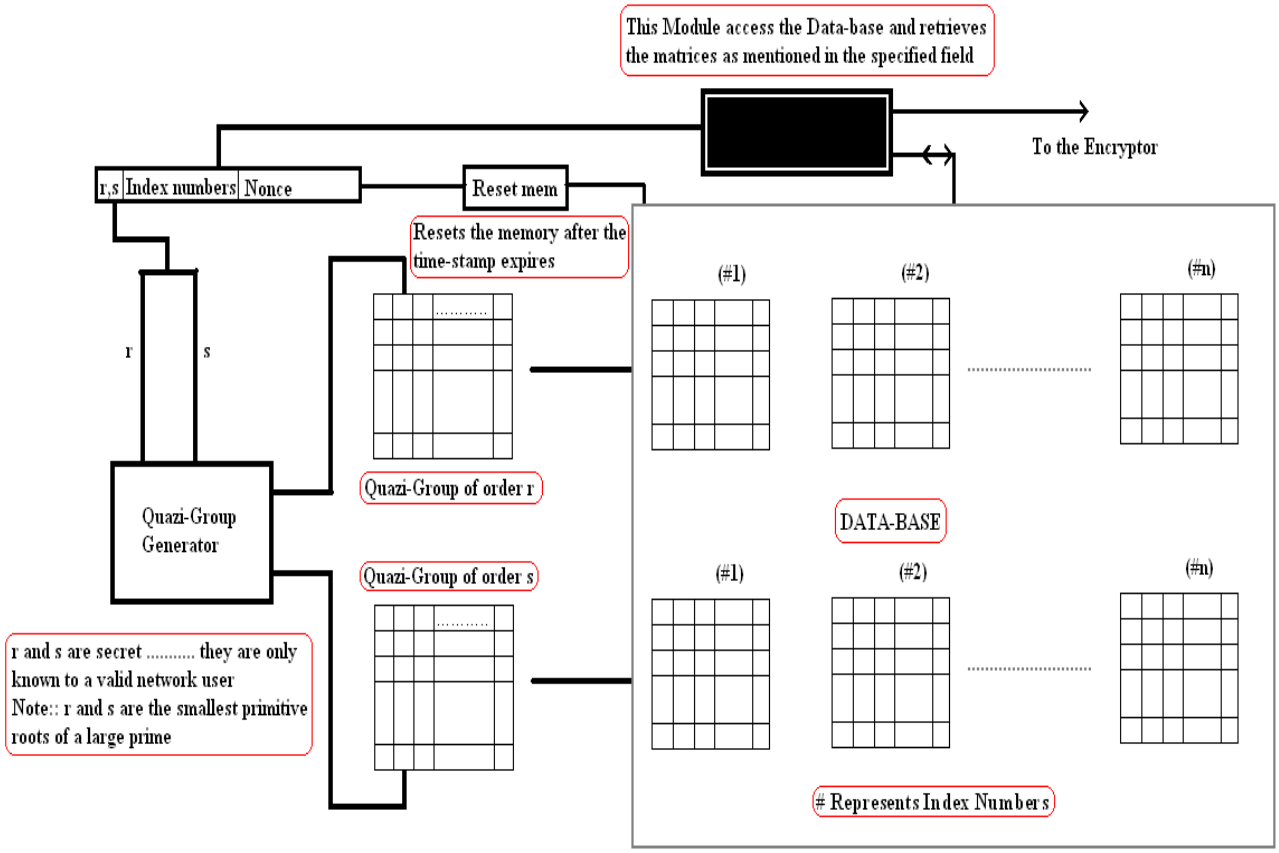


Figure 5: The MIQE Encryptor

MIQE is a randomized algorithm that works on the encrypted key sent by a trusted authority. As illustrated in the above figure (Figure 5), the input to this algorithm is a packet of data that consists of the following fields:

1. The first field consists of the orders of the two quasi groups that have to be generated,
2. The second field consists of the index numbers I_1,.. I_r and I_1…..I_s ,
3. The third field consists of the nonce.

The algorithm first generates a quasi group of order r with all the required properties, followed by the generation of all the possible isotopes of that particular base quasi group. These isotopes are stored in the data base in a specified manner. Later it generates another quasi group of order s and stores the isotopes in a similar fashion.

These quasi groups can be accessed by the encryptor module based upon the index numbers supplied to it. The encryptor module (described in detail in the later sections) codes the incoming data based upon the indices supplied to it; this module is also dependent on the inputs from another module that generates the multiplier seeds q1…qn. The multiplier elements are generated by a special algorithm called MEG 1. It is obvious that all the devices that have to operate in the network have to have this module embedded into their devices. Experimental results show that this algorithm has the capability of scrambling data over a wide range depending upon the index numbers, r, s and the multiplier elements.

4.2 Parameters

In MIQE the final output data is dependent on the following factors:

1. Index numbers
2. Order of the index numbers
3. Order of the matrices *r* and *s*
4. The multiplier elements $(q_1,q_2,q_3,...q_n)$

4.2.1 Index Numbers and Their Order

Perhaps the most important parameters underlying the effectiveness of MIQE, the index numbers may vary depending on the nonce and the order of the matrices. The nonce has two important functions:

1. It is used by the receiver to reset the database after the nonce expires
2. It is used by the Trusted Authority to generate an index and the order of the matrices. This way even the trusted authority does not know the next key that is to be transmitted at any given point of

time. The algorithm that generates the index numbers and the orders of the matrices is named as FG-1 (the frame generator.)

Example 2: Let us assume that the FG1 algorithm generates the values of r, s and the index numbers as shown in the example below. Let us assume the following values:

r=150 and s=270

$I_1, I_2, I_3, I_4, I_5, I_6$:1, 2, 3, 5, 4, 6

Hidden key: 2, 3, 5, 6, 1, 4

Let $d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}$ = 1,1,1,1,1,1,2,2,2,2. Given this amount of structure, this example should show the true scrambling capability of the encryptor.

The output data is represented by the vector: $e_1, e_2, e_3, \ldots e_{10}$

It must be noted that in the example we encode the data using six different matrices and the final output data would have any number in between 1 and 270 (which the order of the second matrix). One might argue that the largest number in the encrypted sequence might be considered as the order of the second matrix, but this is not possible because the encrypted data is produced by mapping the input data through several levels and the probability of the occurrence of the largest number becomes a function of the index vector which is random.

Moreover, experiments have revealed that sufficient security can be provided with group order 'n' as low as 10.

From Table 2 the following deductions may be made:

1. The output sequence does tend to be similar for some sequences of the index numbers, however it does not exhibit any periodicity and is rather unpredictable when we consider the practical case

where the output at any instance is dependent on other constraints such as the order of the matrix involved and the hidden key.

Table 2: The decryption process

Serial Num.	Index Numbers	Output Sequence (e_1,e_2,e_3,…e_{10})
1	1,2,3,5,4,6	126,95,162,16,123,93,163,216,176,249 **
2	1,6,5,4,2,3	125,92,154,267,83,17,30,268,107,14
3	1,4,2,5,6,3	126,95,162,16,123,93,163,216,176,249 **
4	1,5,6,4,2,3	126,95,162,16,123,93,163,216,176,249 **
5	1,4,5,2,6,3	127,99,172,36,158,149,247,66,71,199
6	1,4,6,2,5,3	121,82,137,244,58,268,30,36,209,214
7	2,3,5,6,1,4	270,255,190,9,140,21,91,27,67,169
8	5,6,2,3,1,4	123,84,136,240,62,45,48,22,113,252
9	2,5,3,1,6,4	1,255,189,7,137,17,86,21,60,161 ##
10	2,3,5,1,6,4	1,255,189,7,137,17,86,21,60,161 ##
11	2,3,5,4,6,1	4,261,200,24,161,49,127,72,122,235
12	1,1,1,1,1,1	124,87,139,232,13,161,90,208,152,109
13	2,2,2,2,2,2	2,255,193,18,159,55,146,110,186,63
14	3,3,3,3,3,3	5,259,180,220,231,134,164,31,237,149
15	4,4,4,4,4,4	269,240,131,104,126,62,65,3,113,2

2. One might argue that the eavesdropper can crack this code by simply doing a known plaintext attack. But that is impossible because the algorithm that generates the keys has a random input (owing to the nonce), and this nonce assures that even the Trusted Authority does not know the key that it would be generating in the next instant. Moreover the nonce changes.

4.2.2 Order of Matrices r and s

In general there is no specified restriction on the order of the matrices supplied. However it is mandatory that the order of r be smaller than that of s .One must also note that the final output sequence also depends on the relative difference in the orders of the two matrices. It can be observed that the range of

permissible sequence only depends upon the relative difference in between the order of the first and the order of the second matrix.

4.2.3 The Multiplier Elements

The multiplier elements act like a seed to trigger the encryption process. In order to ensure the reliability of the decrypted data, the whole network must have this key (hidden key). This problem is resolved by embedding a special algorithm called MEG1 into all the valid network devices including the Trusted Authority and the user handsets.

The multiplier elements depend upon the order of the matrix under consideration, the index number, and the nonce. This algorithm ensures the following:

1. It is virtually impossible to break this cipher based on known plaintext attack
2. It reduces the effort to encrypt and decrypt the data for a legitimate user while the computational requirement required to break this cipher is made tremendously high for an illegitimate user.

Chapter 5: MEG1 and FG1 Algorithm

5.1 The FG 1 and MEG 1 Algorithms

The FG-1 and MEG-1 algorithm ensure the cryptographic strength of the MIQE. The Frame Generator or FG-1 is responsible for the generation of the frame that contains the matrix orders, index numbers and nonce to avoid duplication. The MEG1 or Multiplier element generator is responsible for the generation of the multiplier elements that constitute the hidden key

5.2 The FG 1 Algorithm

From our study of the MIQE, it is evident that the security of the entire system depends on the unpredictability of the key (r, s, index elements and nonce).The FG 1 module ensures that even the Trusted authority doesn't know the key that it would be transmitting in the next instant (next time slot).

This algorithm generates the order of the matrices (r,s), index numbers and the nonce, based on a random input from a pseudo-random number generator. This module is essential due to the following reason:

1) This module ensures that the next sequence of indices generated by the system is unpredictable.
2) The nonce is itself dependent on the random input which makes the system more dynamic.
3) The security of the system is dependent on the period of the random number generator and the network time (network time is denoted by T. This time judges the value of nonce *t* that is to be sent in the next frame and at any given instant the value of *t* should not exceed the value of *T*).
4) The security also depends on the other factors like length of the random symbol, number of indices and the order of the quasi groups.

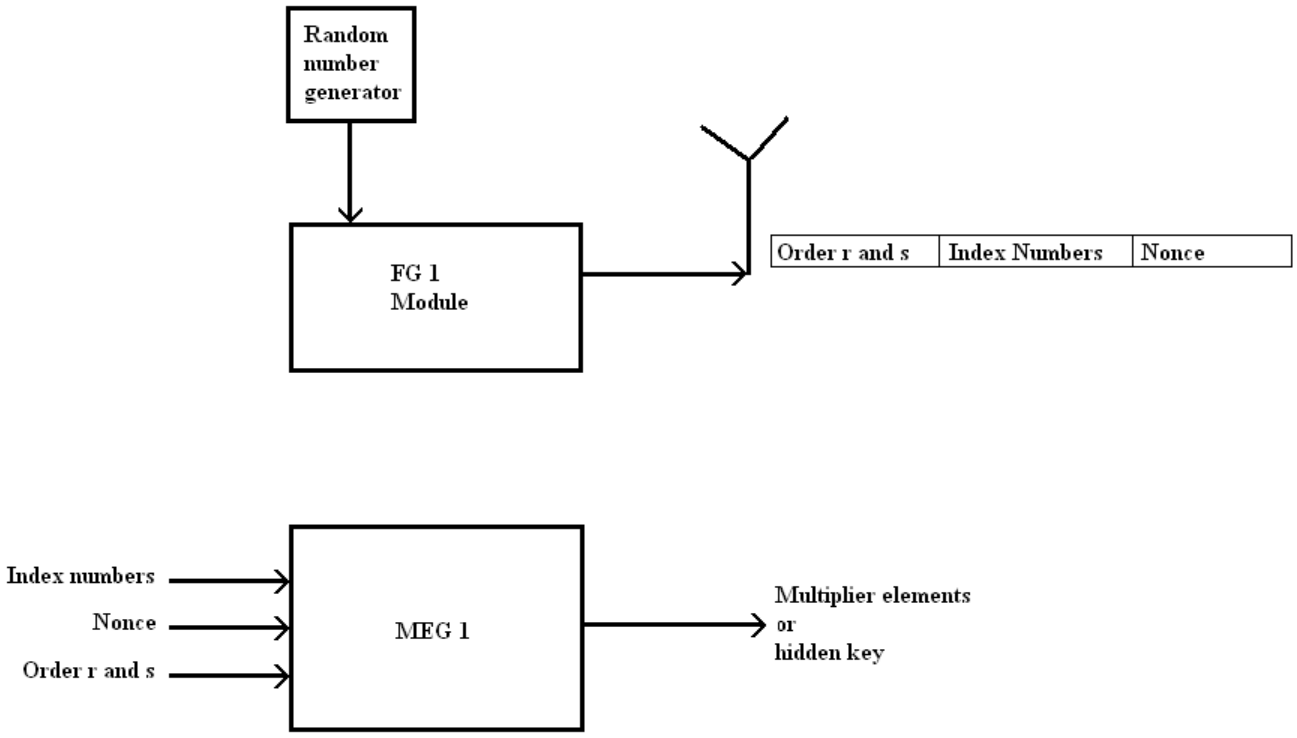


Figure 6: The MEG 1 and FG 1 algorithms

5.3 The MEG1 Algorithm

While the FG 1 algorithm is only required at the trusted authority the MEG 1 algorithm plays a major role in the encryption and the decryption process. During the process of encryption It generates the *hidden key* (explained before) and during the process of decryption the same key has to be generated at the receiver in order to ensure proper decryption. The MEG 1 algorithm takes the entire incoming frame and generates the multiplier elements.

5.3.1 Nonce *t*

The nonce acts like a random input to this algorithm. Since the hidden key generated is dependent on the value of the nonce supplied to it, this algorithm would generate different *hidden keys* even if the same orders (r,s) and index numbers are supplied to it . The nonce might be given by the formula

$$\text{Nonce } t = f(\text{prsg}) \qquad (12)$$

Where the "prsg" stands for the "Pseudo Random Number Generator". We can even use the same random number that is given as the input to the FG-1 module. Since the nonce is generated by FG-1, it is evident that it remains constant for the entire network till it expires and a new transmission is made.

There is a tradeoff of security and overhead involved here. It is required that this nonce changes as frequently as possible, but that increases the computational costs of the entire network. In order to avoid any unnecessary overhead the nonce must be above a certain specified value. Assume that this arbitrary limit is given by T_1:

$$T_1 < \text{Nonce } 't' < T \tag{13}$$

The above equation can be very significant in determining the network overload. If T_1 is very small then the devices in the network would have to recalculate the groups and indices several times per session and on the other hand if T_1 is made too large ($T_1 \rightarrow T$) then there is a high chance of the network being compromised.

5.3.2 Orders r, s

Since in quasi group encryption the multiplier element for a certain level must belong to the quasi group under consideration, the set of multipliers ($q_1, q_2, q_3 \ldots qn$) belongs to the set of quasi group (corresponding indices) that are being used at that particular instant for the process of encryption.

Let us suppose that I_1, I_2, I_3 are the indices corresponding to the quasi group of order r and let I_4, I_5, I_6 be the indices corresponding to the quasi group of order s. Let us suppose that the MEG-1 generates q_1, q_2, q_3, q_4, q_5, q_6. This means that q_1, q_2, q_3 are used as the multiplier elements during the encryption using the quasi group of order r. In other words, for the data to be properly mapped q_1, q_2, q_3 must belong to the group of order r. Likewise, the multipliers q_4, q_5, q_6 must belong to the second group.

Chapter 6: Randomness of the Encrypted Data

6.1 Randomness Introduction

A process or a system is called random if its output follows no specific describable pattern. In general randomness is an objective property, a sequence that seems random to one observer may not appear random to another observer. Mathematically one can generate random sequences that might seem random if the observer has no idea of the algorithm being used, as a matter of fact most of the modern day random number generators are based some mathematical formula.

However the process of using these pseudorandom indices to reference secret matrices generates a totally random signal that bears no resemblance to the original data. Later in this chapter it will be demonstrated that the output is random even if the input is totally predictable (look at example 3). If encrypted data seems random then it becomes very difficult for the eves-dropper to decode the sequence and get intelligible information out of it. Moreover he might think that it is just some channel noise. In chapter 7 we have discussed the application of MIQE to voice scrambling. The output of QVS (Quasi voice scrambler)

6.2 Randomization of Data Using MIQE

In this section we will study the characteristics of the output data by supplying some test data to MIQE. In general the text information sent can be very random, repetitive or have certain characteristics (for example, in English text the probability of occurrence of *e* is maximum) that would make a encoded file vulnerable to the *known plain text attack*. Let us first consider the case of normal English text.

Let us consider the text document of Table 6 is supplied to the MIQE, the characters are then permuted based on the key provided. The scrambling transform used is time dependent and this time

dependency makes it cryptographically strong. The corresponding output data is also shown in the same table. The key used was (35, 41, 5, 4, 2, 1, 6, 3).

Example 3: Input text together with the transformed sequence

- **Input data:**

ONE DAY A COUNTRYMAN GOING TO THE NEST OF HIS GOOSE FOUND THERE AN EGG ALL YELLOW AND GLITTERING WHEN HE TOOK IT UP IT WAS AS HEAVY AS LEAD AND HE WAS GOING TO THROW IT AWAY BECAUSE HE THOUGHT A TRICK HAD BEEN PLAYED UPON HIM BUT HE TOOK IT HOME ON SECOND THOUGHTS AND SOON FOUND TO

- **Encoded data:**

23 32 17 38 35 8 12 21 24 39 1 26 6 39 15 1 3 6 39 34 4 11 40 29 4 19 20 2 9 34 20 5 6 29 37 6 5 41 23 17 29 37 16 30 28 33 13 41 30 18 11 40 40 19 40 13 23 21 30 34 27 40 8 37 8 14 5 15 23 20 30 32 35 24 26 34 21 39 41 14 4 5 16 26 37 33 11 3 37 34 8 12 33 16 18 36 2 26 12 29 22 36 27 16 18 25 17 19 8 8 33 6 28 26 17 39 19 33 22 38 34 32 27 12 4 19 5 29 16 33 33 39 34 37 38 36 24 25 5 14 39 3 38 13 20 30 30 26 37 31 17 8 3 19 8 33 17 34 19 27 25 6 19 30 19 19 7 10 1 30 20 3 29 6 40 33 19 13 36 39 35 25 30 22 34 16 18 13 31 19 28 4 28 9 39 14 24 4 6 22 10 39 32 21 15 37 19 19 4 29 13 26 4 16 6 29 18 29 7 21 19 19 16 38 19 9 37 6 16 14 25 22 26 14 19 34 11 13 3 13 27 27 17 9 17 34 30 3 9 15 34 15 3 11 22 18 39 26 19 41 32 29 10 20 11 11 13 35 40 39 30 24 10 12 17 3 28 41 26 5 32 19 19

Key used: 35, 41,5,4,2,1,6,3

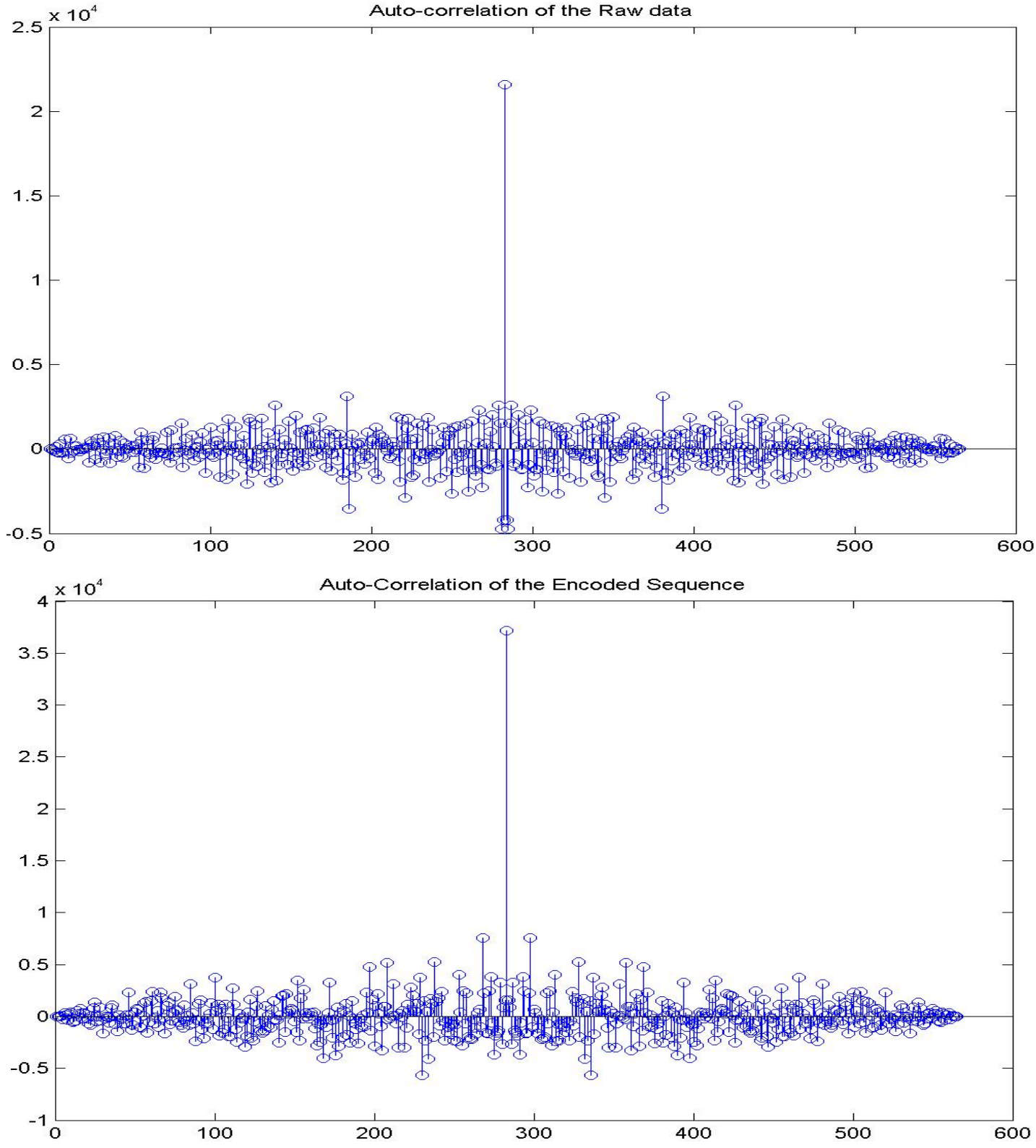


Figure 7: Autocorrelation of the input data and the encoded data for Case 1

Example 4

Let us suppose that the data supplied to the MIQE is highly redundant and it consists of a string of Es. This example would illustrate its scrambling capabilities. In spite of this the output is mapped to a range of values from 1 to 41. It must be noted that the first 2 integers in the key are the order of the matrices and the remaining are the encryption indices.

- **Input data:**

E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E

E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E

E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E

E E E

- **Encoded data:**

11 10 24 28 33 11 23 8 28 10 17 20 2 40 29 15 22 2 30 6 30 7 29 37 22 10 7 9 18 4

29 38 12 30 17 33 7 10 27 28 38 38 33 12 11 19 4 12 29 6 7 12 9 37 16 12 34 26 18 5

40 17 39 36 41 38 36 16 31 34 8 32 35 9 33 1 34 29 20 1 33 7 33 34 33 13 9 5 38 3

39 38 6 32 18 34 16 21 34 9 41 24 12 34 30 35 19 7 32 7 32 1 24 21 20 5 27 11 23 7

12 27 35 12 9 18 2 33 41 9 33 32 23 35 5 29 15 16 23 5 21 40 3 6 16 4 27 18 23 12

30 1 27 15 19 17 22 18 35 23 38 27 36 18 35 24 17 27 6 9 23 37 32 36 23 41 6 5 28 20

26 2 6 6 33 32 8 21 3 12 1 32 29 22 6 26 21 4 33 7 38 33 16 23 5 8 12 21 19 40 6

24 1 28 9 4 5 7 37 2 35 8 23 4 16 7 3 12 22 18 36 27 31 37 19 34 35 7 17 8 36 9 34

21 30 28 7 27 40 35 25 16 19 10 15 20 6 8 18 1 22 8 34 22 25 36 19 32 40 29 9 35 34

10 35 41 34 35 11 17 23 11 36 35 22 25 34 38 14 25 39 12 34 27 26 35 26 20 27 1

Key used: 35, 41, 5, 4, 2, 1, 6, 3

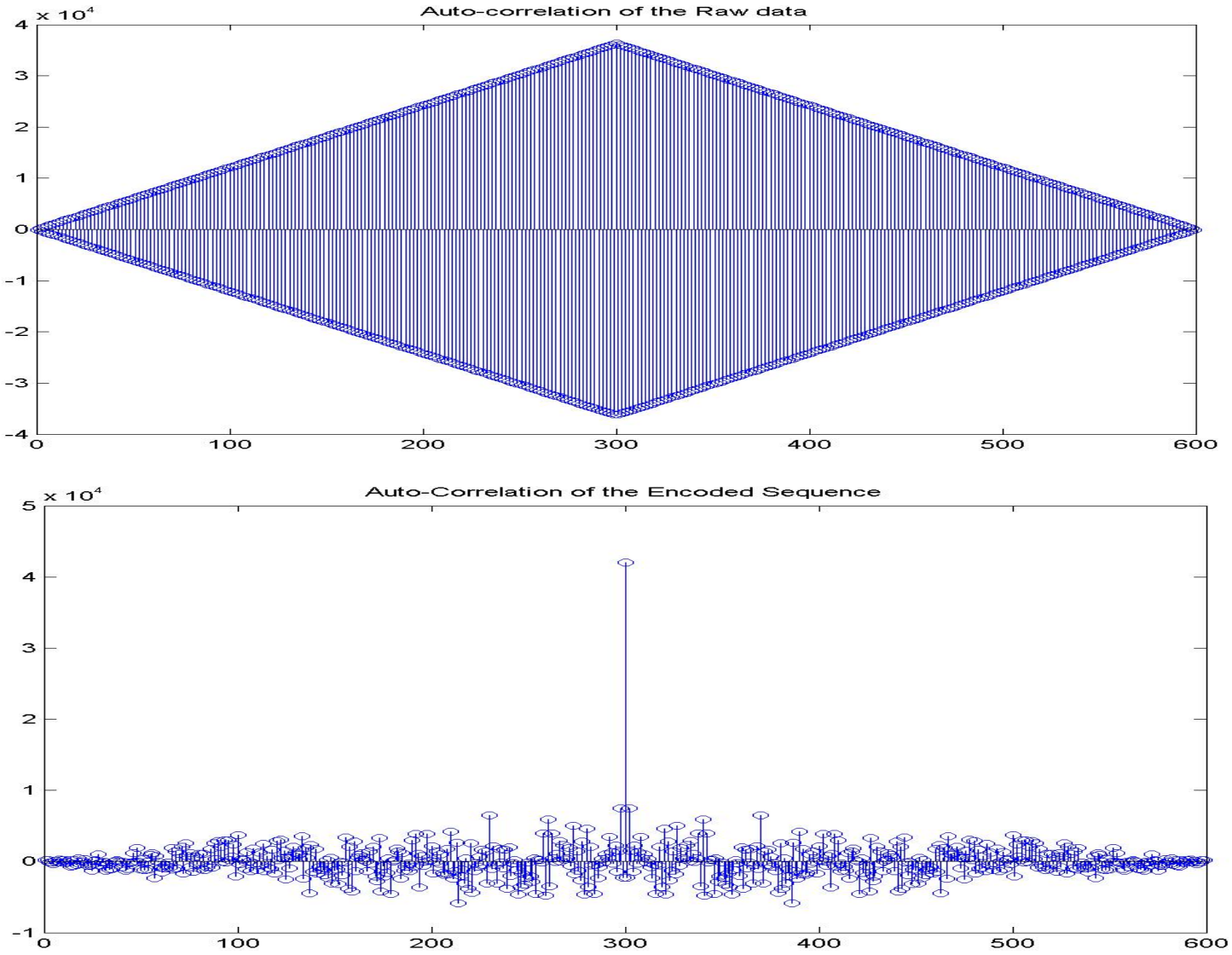


Figure 8. Autocorrelation of the raw data and the encoded data for a constant input sequence

We see from the autocorrelation graphs that the encrypted sequence is essentially two-valued and, therefore, the output sequence is very random.

In the above three examples it can it can be observed that the output is random even when the input is highly predictable. This is because the quasi group encryption is highly time dependent. This time dependency has its own advantages and disadvantages. Since it is time dependent the described cryptosystem can be implemented in parallel structure which would further enhance the productivity of the system. This would however make the system more susceptible to errors that might occur during

normal transmission. These errors can be reduced by operating the MIQE In block mode. In this mode chunks of data is gathered (say each chunk containing about a 1000 symbols) and is passed through the encryptor. Now, if the received data is corrupted then the effect of error is localized to that particular block.

Chapter 7: QVS – Quasi Group Voice Scrambler

7.1 Background

There are a several ways to implement a speech encryption. Some suggest that speech can be encrypted using masking techniques. However these masking techniques cause increase in the bandwidth which is not desirable, since in it increases the network overhead. On the other hand some authors have suggested that scrambling might be a better approach, since a scrambled speech does not require any extra transmission bandwidth. The QVS falls in to the second category.

Let us revisit the scrambling techniques. The following diagram depicts the two possible ways of speech scrambling (Figure 9)

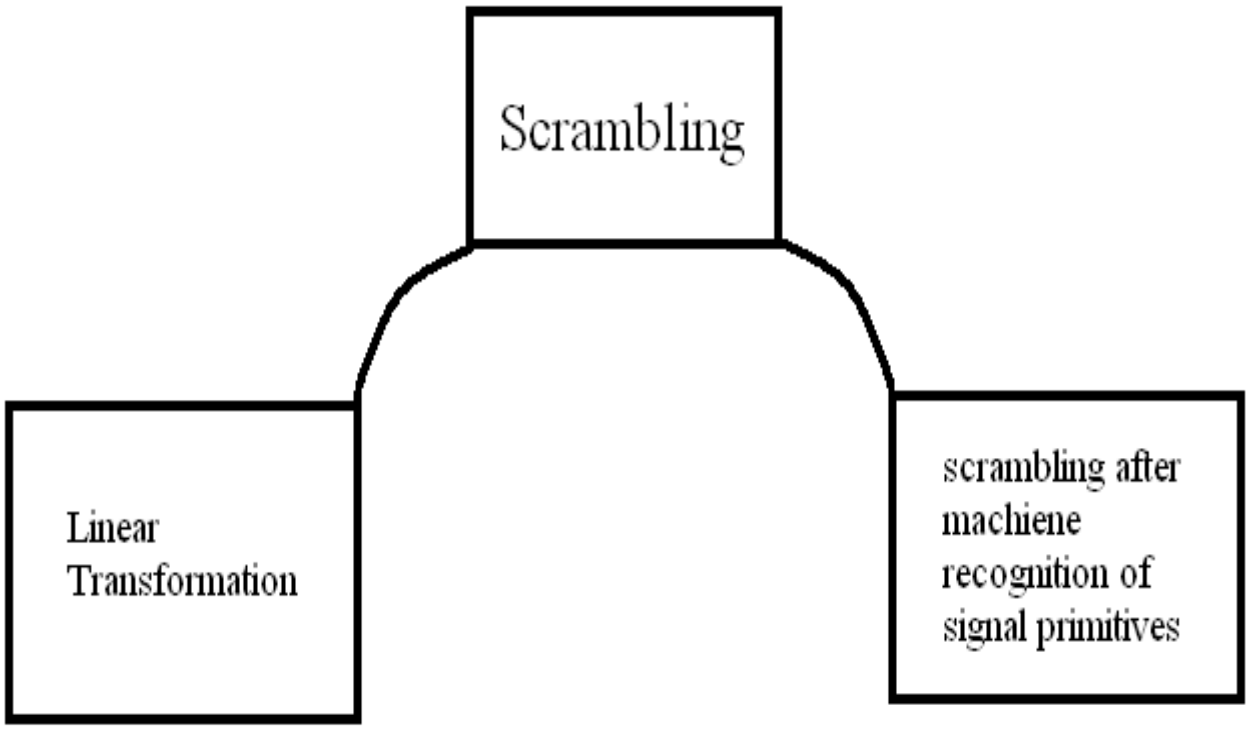
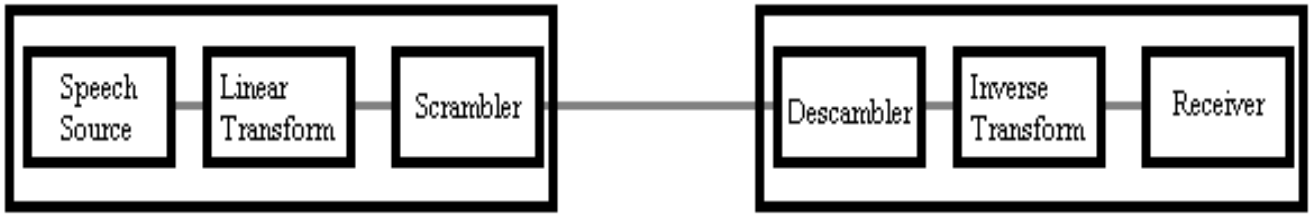
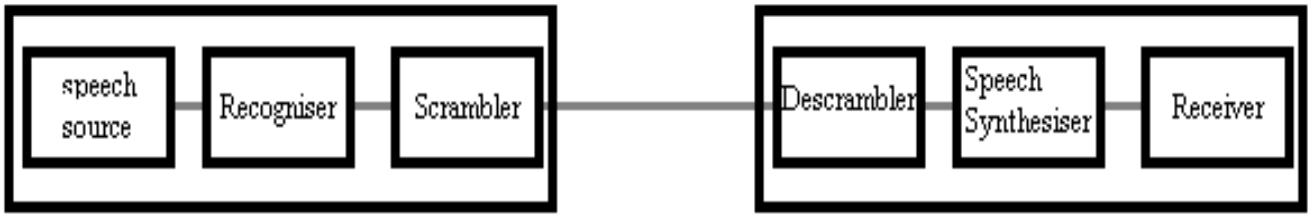


Figure 9: two types of scrambling

In the first method the speech source is sampled and a linear transform is applied to it. The transformed speech is then scrambled. At the destination the received speech is descrambled and inverse transform is applied on it to retrieve the data. In order to decipher the signal the illegitimate user has to first find out the method used for scrambling the data. If he somehow manages to discover the underlying scrambling transform he might still find it difficult to find the linear transform. However if either the linear transform or the scrambling transform is kept a secret [18] then it would be more difficult to crack the system. Thus if one takes sufficient time to analyze network traffic it wouldn't be difficult to break this cipher. This system is highly susceptible to known plaintext attack (since the transform is static). To make this system cryptographically strong one has to either keep changing the linear transform or send different keys to the scrambler; so that it increases the number of possibilities



Scrambling following a linear transformation



Scrambling after machine recognition of signal primitives

Figure 10: block diagrams of the two scrambling techniques

In the second approach the speech is decomposed into its primitives. Symbols are assigned to each primitive and these symbols are scrambled. Though this system sounds simple, it needs advanced speech recognition module that is capable or recognizing speech primitives and assign unique symbols to each primitive. Unfortunately due to the large number of possibilities it is very difficult to separate primitives. More over speech synthesis generally produces somewhat mechanically sounding speech. So we need complex signal processing algorithms to make this system usable. Due to the complexity involved the second method is generally not used. The QVS system first samples data and performs compression on it. In this stage the voice signal is generally decimated such that no significant information is lost. The compressed signal under goes some preprocessing to make it more presentable to the built in MIQE algorithm. The MIQE algorithm then transforms the give signal in to a highly random signal that resembles noise in its characteristics. Let us now go through the QVS protocol in detail

7.2 QVS – Quasi Group Voice Scrambler

This section describes an application of MIQE to speech scrambling [19]. Since the sampled speech is not of integer type, some sort of pre-processing must be done because MIQE only scrambles integer data (or binary). The quasi group voice scrambler (QVS) constitutes of an encoder and a decoder (Figure 11 is the encoder-decoder pair).

The encoder as indicated in the illustration has a pre-processor and MIQE while the decoder has MIQE and post-processor. It should be noted that the QVS algorithm is working on a block of data. To achieve optimum performance one has to converge at an optimum block size. Small block sizes correspond to faster operation and lower error rates while large block sizes correspond to rapid information transfer capability and higher bandwidth requirement. We conducted experiments tacking block sizes of 200, 512, 1020, 15000 (there is no underlying principle that dictated the block size). From the observations

made it was clear that higher block sizes meant longer encryption time. Speed can be improved by using parallel any stand parallel processing techniques.

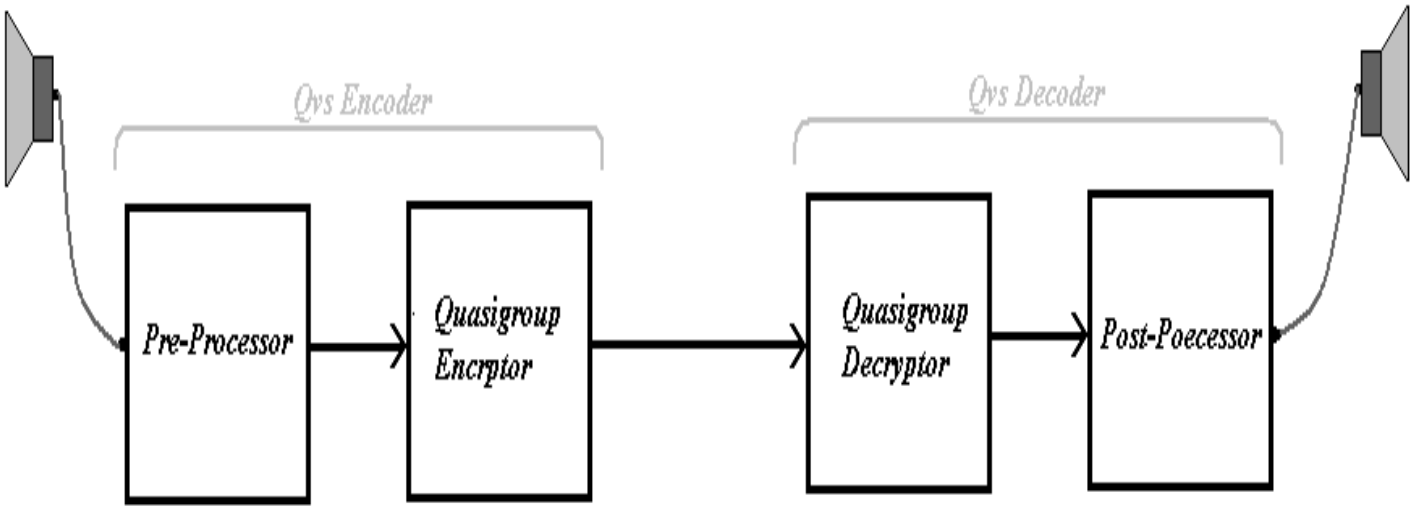


Figure 11: QVS block diagram.

7.3 The Pre-processor

This subsystem (Figure 10) consists of a sample and hold circuit, a decimator of a factor r and a pre-transforming function $Q_{in}(d)$, where d represents the raw data. After the speech signal is sampled, it is passed through a decimator. This step is essential to reduce the redundancy that is inherent in speech data. The third step is perhaps the most important one. The transform $Q_{in}(d)$ takes the sample values of the speech signal and converts them to integer values without sacrificing the voice clarity.

It must be noted that for telephone quality speech we need to be concerned with frequencies between 300 Hz to 3.6 kHz. Thus we also need a band pass filter to pass only the frequency of interest. However we assume that $S_a(t)$ is an NB analog-signal.

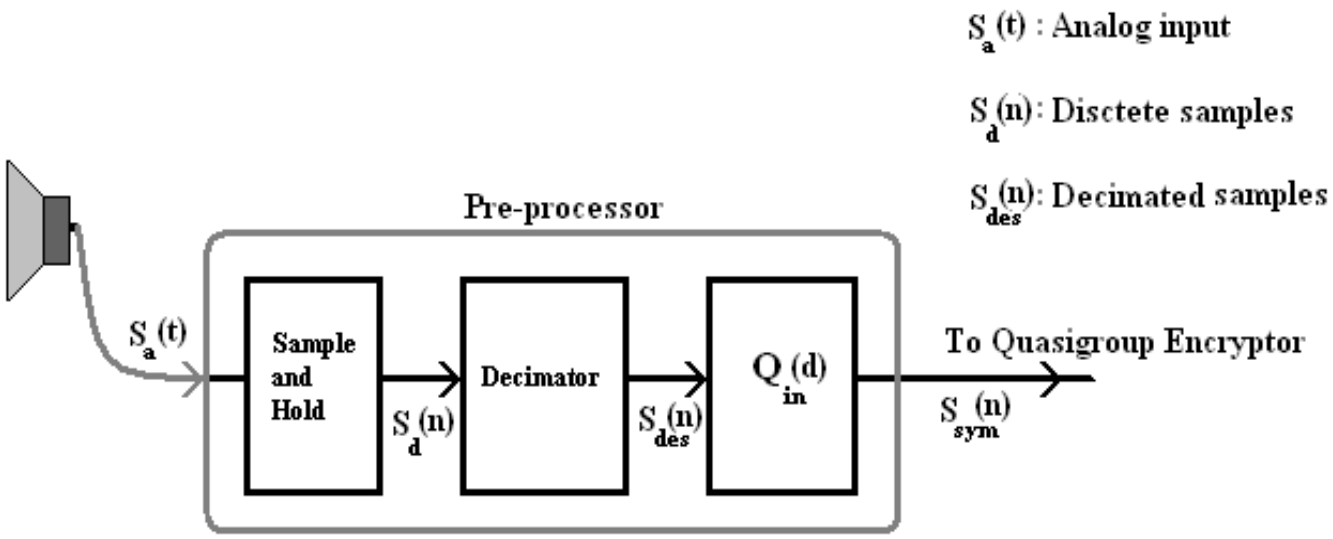


Figure 12: Block diagram of the Pre-processor

The output of the pre-processor are integer symbols representing the discrete samples $S_{des}(n)$.There is yet another restriction on the largest symbol that is used by the pre-processor to represent an input sample. If the incoming sample is 857 then we need a quasi group of the size 857 to effectively encode the symbol. Creating a quasi group of that size could take considerable time and resources. By keeping the symbol range to a controllably small value, one can prevent heavy costs and maximize the speed and efficiency of the encoder.

To make the encoder more efficient we can convert the output of the pre processor in to lower range of symbols. We can do this by first getting the binary equivalent or a certain symbol and then splitting the code-word in to two halves. Later this code-word is converted in to two symbols this way if the output symbol is 8 bits code word is converted to two 4 bit code words(there by reducing the dynamic range from 0-256 to 0-16). And it is not computationally intensive to generate a 16 by16 quasi group. We can also split 8 bit code word to four two bit code words and further reduce the dynamic range but one must note that while this splitting reduces the size of the quasi group that has to be generated it also increases the number of encryptions/decryptions that have to be performed.

We need to make a choice between the size of the quasi group generated and the number of encryption/decryptions cycles to ensure quality, speed and complexity. If a

7.4 Experimental Results: Pre-processor

In Figure 11 the first graph (a) is the output of the pre-processor. We can see that it is periodic from its auto-correlation properties. However after scrambling the output becomes highly randomized (the auto-correlation of the encoded data is same as that of a random sequence). Hearing tests show that the encoded sequence sounds similar to the sequence generated by the 'rand' function (in Matlab). The decoded sequence on the other hand sounds exactly the same as that of the original signal.

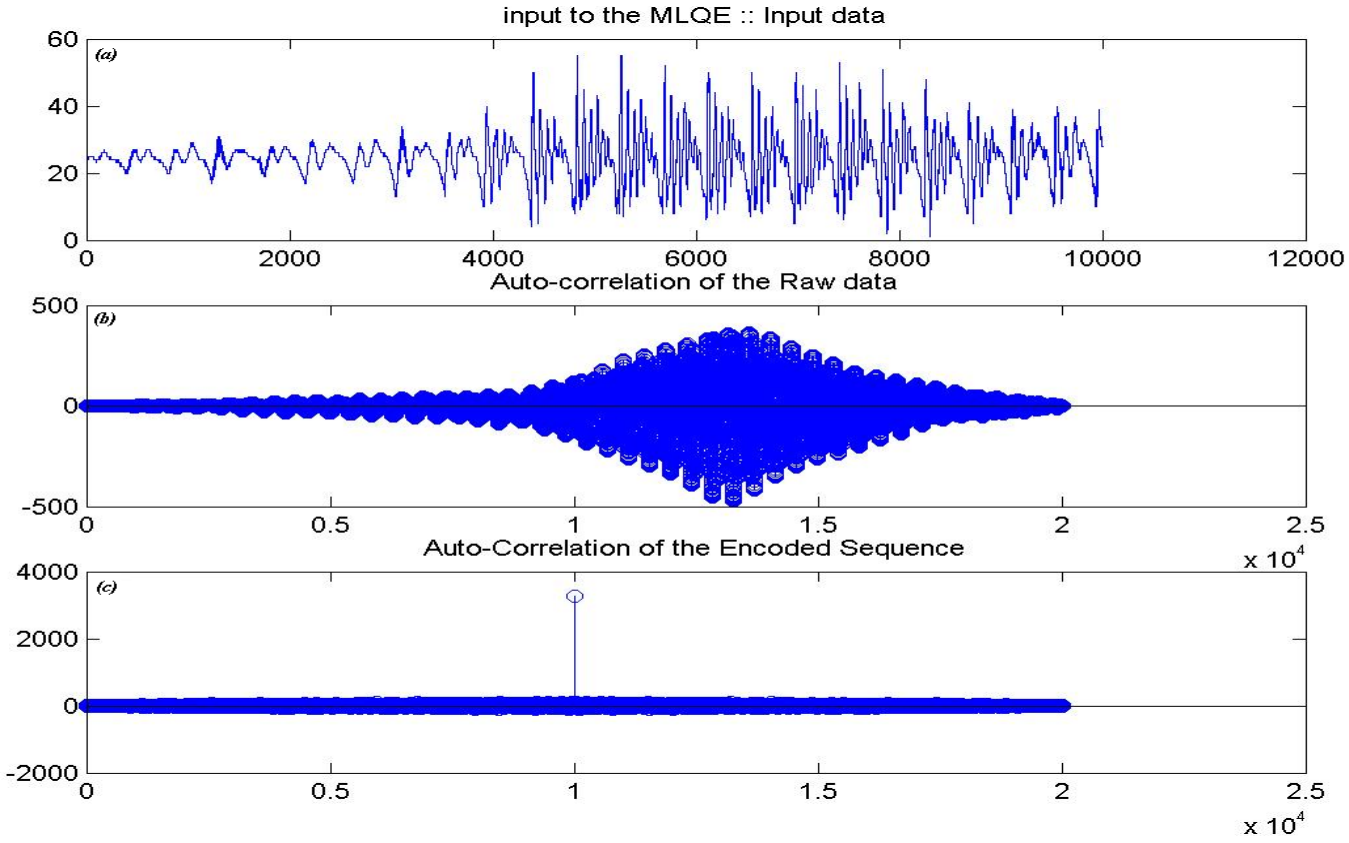


Figure 13: Speech input and autocorrelation of input and output

7.5 The Post-processor

The post processor does exactly the reverse operation of the pre-processor. The input to the post-processor is the decoded symbol vector $S_{sym}(n)$ from the MIQE. This sequence of symbols is transformed to the original form by the inverse transform $Q^{-1}(e)$ (where e is the decoded symbol vector). The discrete set of samples are interpolated (the discrete set of samples is represented as *d* in Figure 12) to obtain the analog signal that was initially transmitted. (This step is only meant to improve the speech clarity). If the pre-processor decimates the original samples by a factor *r* then the post-processor interpolates the sequence by the same factor *r*. This is done to counter the loss of speech information caused by the decimator. It would be difficult to interpret the decoded data without the knowledge of the inverse transform.

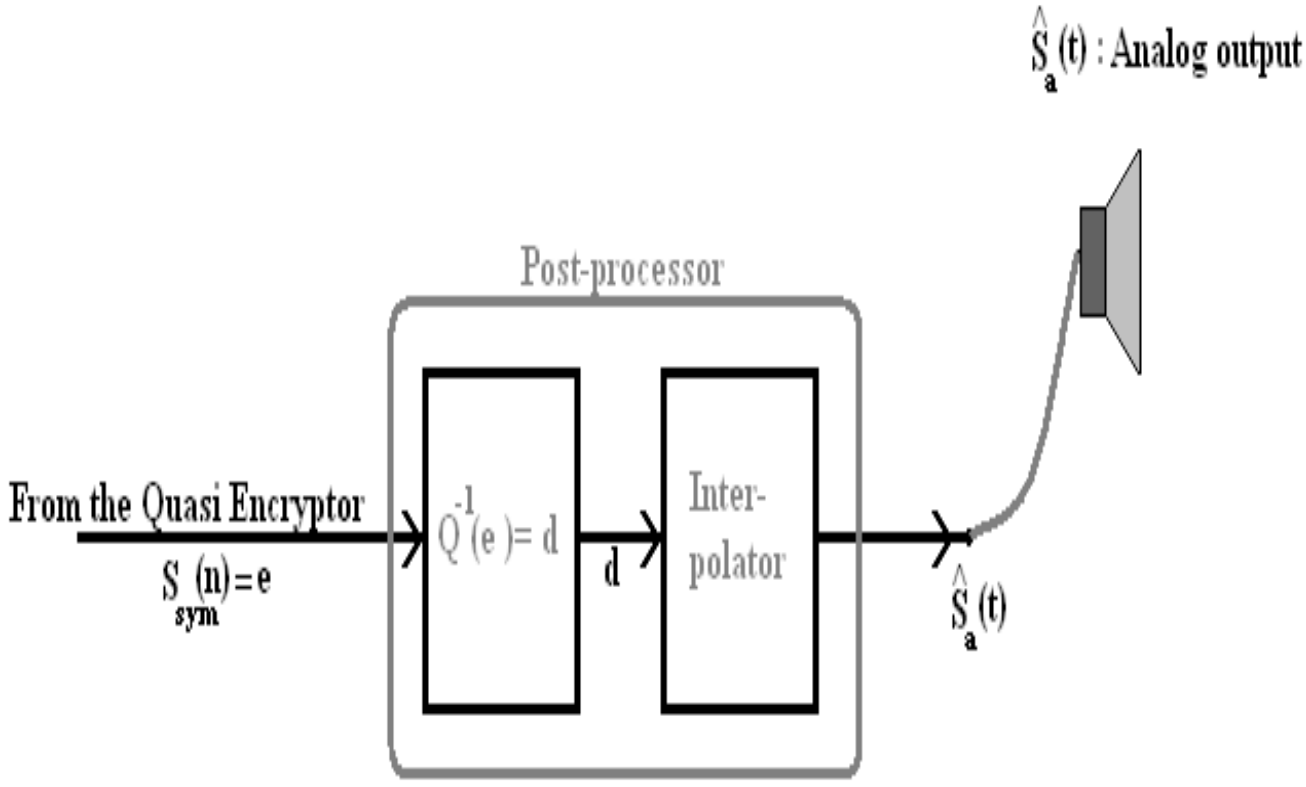


Figure 14: Block diagram of a Post-processor

7.6 Experimental Results: Post-processor

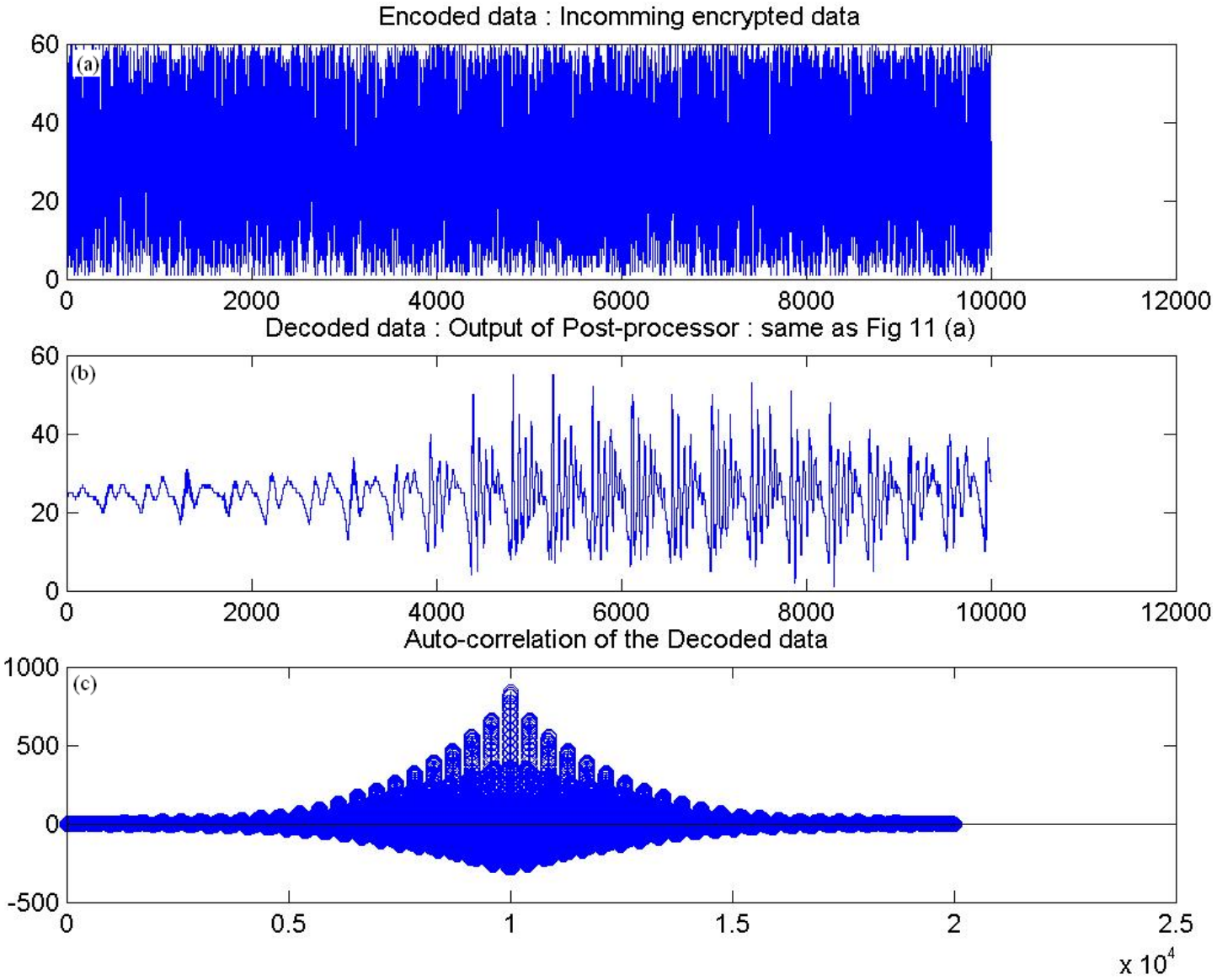


Figure 15: Graphs of the encoded data, decoded: Post-processor output and autocorrelation of decoded data

It may be observed that the encoded sequence is highly randomized. However the decoded sequence is similar to the transmitted sequence. The autocorrelation property of the decoded sequence (Figure 13 c) is slightly different than the autocorrelation of the input data; this is because of the mismatch in the

interpolation factor r_1 and the decimation factor r_2. It must be noted that the autocorrelation graphs cannot be identical because of the inherent loss of information in the transmitted data. Listening tests have revealed that the output of the encoder sounds similar to noise. This happens even when no information is sent. Thus the is no way of telling when the transmission has stared/ended. Since this transform is time dependent it becomes all the more difficult to distinguish between the two.

7.7 Application of QVS In Cellular Networks

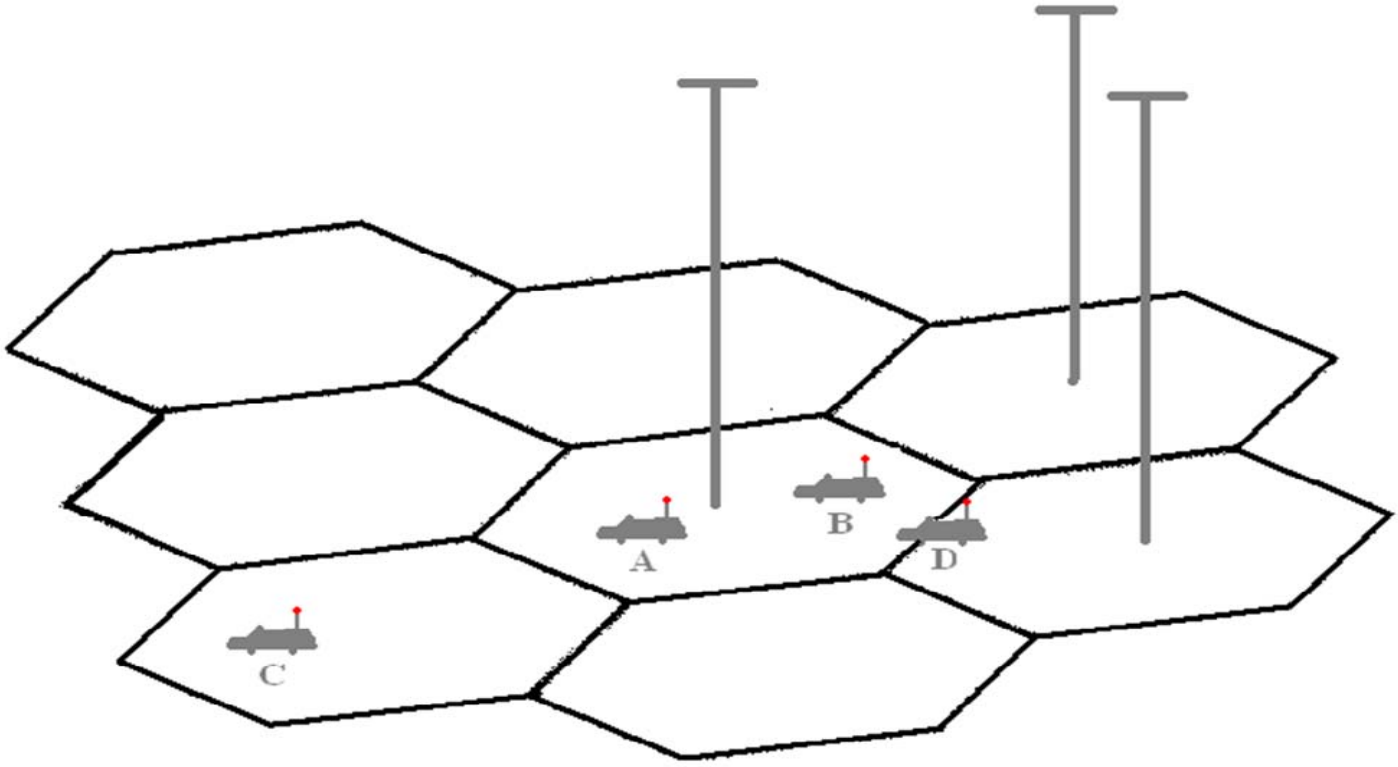


Figure 16: a possible application of QVS to the cellular network

Generally, a cellular network consists of hexagonal coverage regions called cells. Each of the coverage regions has a central antenna called cell site. The cell site is responsible for handling all the

users in a specific coverage area that is under its coverage. Apart from this it has several responsibilities. Let us first list the duties of the cell site:

1. It has to connect two users when demanded
2. It has to keep track of all the users in its coverage area.
3. It has to allow new users that have entered its coverage area from its adjacent cell.
4. It has to notify the appropriate cell when a user is leaving its coverage area and entering the new coverage area.

In the normal operation when the user A wants to talk to B then he requests the cell site to establish a connection to B. So the cell site has to determine the location of B; in this case B is located in the same cell site as A. Thus the cell site can directly allocate the channel between A and B. Now let us consider the second case where A wants to communicate with C. The cell site refers to the Global position register and finds that the user c is in the adjacent cell. It contacts the corresponding cell site and informs it that the user A wants to communicate with user B. The two cell sites acknowledge and open a communication channel between A and B. It must be noted that in both cases that the connection is made through the cell site. We now have to discuss the role the QVS in this scheme. After the connection is made, if the user A wants to send a secret code to B he has to request the cell site for a secured encrypted connection. Then the cell site authenticates the two parties and enables the encryption module that is embedded on all the devices on the network. The cell site also has to supply the two parties with the index numbers and orders that are used for encryption. This way the index numbers and orders are unique to a particular user pair. Shortage of keys will not be experienced since FG 1 can generate many index vectors depending on the order used and number of isotope allowed. We have already shown that the change in the order of index numbers also affects the output of the encoder. This property also increases the number of possible keys that can be generated. FG1 is only present in the cell

sites where as MEG 1 is present in both cell site and the end user device. FG1 can be called a secret hardware [18] since the actual process of generating the keys is kept from the users.

The load on FG1 is of some concern. It should be noted that the FG1 comes to action only if two calling parties demand a secured connection. When this happens the FG1 algorithm first sends an activation signal to the two parties to activate the MEG1 module. Then the two parties agree upon a certain criteria. Based upon the agreed upon criteria the FG1 algorithm generates the group orders and number of isotopes and indices. This information is transmitted to the two parties. Ones this is done the parties can start exchanging information.

Since the cell site has to now distribute criteria and encryption keys, certain amount of overhead is inevitable. The load on FG1 depends on the number of user pairs demanding secure communication.

Chapter 8: Security Analysis

8.1 Security of MLQE

A loop is a quasi group with an identity element e:

$$x*e=x=e*x \tag{14}$$

If one could find a loop in a quasi group transformation that would facilitate the task of breaking it. But determining this is not easy and this task will equal the number of transformations associated with the quasi group. Since a quasi group is defined by its multiplication table which is a Latin square, its security would depend on the number of such Latin squares. The eavesdropper will have to guess the specific Latin square used out of the total number that is possible.

A Latin square of order n is an n × n array in which n^2 symbols, taken from a set A, are arranged so that each symbol occurs only once in each row and exactly once in each column. A Latin square is said to be normalized or reduced if the elements of both its first row and its first column are in a natural order. For n ≥ 2, the total number LS(n) of Latin squares of order n is given by

$$LS(n) = n!\,(n-1)!\,T(n) \tag{15}$$

Where T(n) denotes the number of reduced Latin squares of order n. The numbers T(n) and LS(n) increase very quickly with n. Tables 3-5 give the number of reduced Latin squares (see [8] for details), where for large n, the number of Latin squares is estimated using the double bound:

$$\prod_{k=1}^{n}(k!)^{\frac{n}{k}} \geq LS(n) \geq \frac{(n!)^{2n}}{n^{n^2}}. \tag{16}$$

Table 3. Reduced Latin squares for $2 \leq n \leq 10$

N	T(n)
2	1
3	1
4	4
5	56
6	9,048
7	16,942,808
8	535,281,401,585
9	377,597,570,964,258,816
10	7,580,721,160,132,811,489,280

Table 4. Reduced Latin squares for n == 11, 12, 13, 14, 15

N	T(n)
11	$5.35 \,.\, 10^{33}$
12	$1.62 \,.\, 10^{44}$
13	$2.51 \,.\, 10^{56}$
14	$2.33 \,.\, 10^{70}$
15	$1.50 \,.\, 10^{86}$

Table 5. Bounds on the number of Latin squares for n = 16, 32, 64, 128, 256

N	Lower Limit (number of Latin squares)	Upper Limit (number of Latin squares)
16	$.101*10^{119}$	$.689*10^{138}$
32	$.414*10^{726}$	$.985*10^{784}$
64	$.133*10^{4008}$	$.176*10^{4169}$
128	$.337*10^{20666}$	$.164*10^{21091}$
256	$.304*10^{101724}$	$.753*10^{102805}$

It is clear from these tables that the task of breaking the quasi group cipher is of astronomical complexity. Thus if the key is temporary, it would be very difficult to extract the information using brute

force. The known plain text attack and replay attack are also not possible because the key keeps changing.

Chapter 9: Conclusions and Future Work

We have shown that quasi group scrambling constitutes an excellent method of encryption and generation of pseudo-random sequences. Quasi groups (or Latin squares) provide a powerful method for generating a large set of permutation transformations by permuting not only the samples but also transforming the amplitudes themselves across their range. We have also implemented a Quasi Voice Scrambler which is capable of converting a voice signal to a random noise.

The proposed MIQE algorithm could use several group orders (in our examples we use two) and in our example we use six indices. If we assume that these indices are in the range of 1 to 8 (as in the example) and the maximum group order 'n' is 25, the order requires 32 bit and the indices require 3 bit each, which sums to 18 (if we consider 6 indices) bits. So the key length excluding nonce and time stamp is 32+18 = 50 bits. If the eavesdropper has to use brute force he simply has to try all 2^{50} combinations. However the key size does not include the nonce and time stamp. Moreover since the key constitutes of individual fields it is more difficult to guess the numbers in these fields. Even if one guesses the indices he still has to figure out the hidden key which makes the number of possibilities to $2^{50} * 2^{18}$ which is quite large and since it is continuous the beginning is also unknown. Thus the proposed example system can compete with DES since the complexity involved in breaking DES is simply 2^{64} possibilities for a key size of 64 bit. The RSA on the other hand provides greater security than this for the typical values that are currently used. However, we can always use appropriate length keys and index numbers and the value of nonce to obtain security that exceeds that of DES as well as RSA or the more recently introduced ECC cryptosystems.

As further research we propose finding efficient ways to send the indices such that the eavesdropper finds it difficult to extract the actual numbers. This however is not a critical problem for the system as the key is updated randomly based on the time stamp.

References

[1] Bailey, R.A., Cameron, P.J. 2003. : Latin squares: Equivalents and equivalence August 1, 2003

[2] Bakhtiari, S., Safavi-Naini, R., Pieprzyk, J. 1997. A Message Authentication Code Based on Latin Squares, Proc. Australasian Conference on Information Security and Privacy, pp. 194–203, 1997.

[3] Feistel, H. 1973. Cryptography and computer security. Scientific American, May 1973.

[4] Gligoroski, D. 2004. Stream cipher based on quasi group string transformations in Z_p^*. [arXiv: cs.CR/0403043]

[5] Gligoroski, D. 2005. Candidate One-Way Functions and One-Way Permutations Based on Quasi group String Transformations. arXiv:cs.CR/0510018

[6] Kak, S. and Jayant, N.S. 1977. Speech encryption using waveform scrambling. Bell System Technical Journal, vol. 56, pp. 781-808.

[7] Kak, S. 1983. Overview of analogue signal encryption. IEE Proceedings F. Communications, Radar and Signal Processing. vol. 130, pp. 399-404.

[8] Kościenly, C. 2002. Generating quasi groups for cryptographic applications. Int. J. Appl. Math. Comput. Sci., vol.12, No.4, 559–569.

[9] S. Markovski, S., Gligoroski, D., Bakeva, V. 1999. Quasi group string processing: Part 1, Proc. of Maced. Acad. of Sci. and Arts for Math. and Tech. Sci., XX 1-2, 13–28.

[10] Markovski, S. and Kusakatov, V. 2000. Quasi group string processing: Part 2, Proc. of Maced. Acad. of Sci. and Arts for Math. and Tech. Sci., XXI, 1-2, 15–32.

[11] Ritter, T. 1998. Orthogonal Latin Squares, Nonlinear Balanced Block Mixers. Ritter Software Engineering Report.

[12] Schnorr, C.P. and Vaudenay, S. 1995, Black box cryptanalysis of hash networks based on multipermutations, Lecture Notes in Computer Science, 950, pp. 47–57.

[13] Vaudenay, S. 1994. On the need for multipermutations: Cryptanalysis of MD4 and SAFER, Proc. Fast Software Encryption, pp 286–297.

[14] S. Kak, “Information complexity of quantum gates." International Journal of Theoretical Physics, vol. 45, pp. 933-941, 2006.

[15] S. Kak, “On the realizability of quantum computers." ACM Ubiquity, vol. 7 (11), pp. 1-9, 2006.

[16] S. Kak, “Statistical constraints in starting a quantum computation." Pramana, Journal of

Physics, vol. 57, pp. 683-688, 2001.

[17] S. Kak, “An overview of analog encryption," Proceedings IEE, vol. 130, Pt. F, pp. 399-404, August 1983.

[18] S. Kak, “On secret hardware, public-key cryptography," Computers and Digital Technique (Proc. IEE - Part E), vol. 133, pp. 94-96, 1986.

[19] S.kak, “Scrambling and Randomization,” Technical report EE 606, June 4, 1981.

[20] S. Singh, “The Code Book: The science of Secrecy From Ancient Egypt To Quantum Cryptography” Anchor Books, New York, 1999.

Vita

Maruti Venkat Kartik Satti received his bachelor's degree (B>Tech) in electronics and communications in the year 2005 from Jawaharlal Nehru Technological University, Hyderabad, India. His zeal in engineering course has made him pursue a master's degree in electrical engineering at LSU. AS a B.tech student he has worked on a very interesting topic entitled "**Data Scrambling using Galois polynomial"** in which he developed the technique of encryption using random polynomials. The aptitude for this area made him opt for the courses like Cryptography DSP, data communications and several other interesting subjects in the master's program. Under the able guidance of his professor Dr Subash Kak he has completed his masters thesis on "Quasi group theory" .This work has been published in the archives. He is the co inventor of the patency on "Voice Scrambler" which is being filed by Dr Subash Kak at LSU

Born in the 80's as a second child of educated parents (father is an attorney and mother is a professor in genetics), Mr Satti is highly intellectual, analytical and motivated. He has very high moral values and holds extreme respect towards his teachers.

He feels that LSU has taught him a lot that would help him to survive .His hobbies include music, computer games and modal making specially ships.